25.12.2020

Emre Can Şen | 21902516

Section-1

EEE 102 –Term Project Lab Report

Miner Game

**Youtube Video Link**

https://www.youtube.com/watch?v=2TH5cndzHOA&t=2s&ab\_channel=EmreCanSen

**Abstract**

Objective of my term project was to design a basic video game that was inspired by the video game “Gold Miner”. In the game player simply controls the crane by pressing the button, which. Player’s aim is to correctly time the button presses in order to catch the moving stones. Randomly generated stones have varying attributes and players must keep in mind these attributes and varying effects of these on the gameplay to catch more stones to gain higher scores before letting pass too much stones and game ending. I used a Basys3 and VHDL programing language to do the project.

**Project’s Design Specification Plan**

**Mechanics of the Game**

Initially, I had the idea to implement a version of the video game “Gold Miner”, in which you control a crane to catch the stone pieces which provide points. However upon facing design setbacks, I have decided to change the certain aspects of the design while keeping the main goal of the game. Game’s control scheme and what is on the screen are very simple. There is a crane that player controls which is mounted stationary on the horizontal middle of the screen and variety of stationary stones are placed within the reach of crane’s range. Crane starts to swing about 60 degrees from the vertical line and after that comes back to vertical line and does another 60 degrees on the other side.



Figure 1: Gold Miner Initial

,



Figure 2: Gold Miner Crane Draw

Crane continues to repeat this motion. Player is responsible for correctly timing the button press. Upon pressing the button, crane will move towards where it was pointing at the time of the button press. For example, if the degree with the vertical line and the crane was 30 degrees, crane would start the motion and follow a straight line while keeping that 30 degrees. If the crane does not collide with any of the stones, it will bounce back from the edges of the screen and follow back the same path that it came from, while keeping its speed constant throughout the motion. If the crane was to collide with a stone, crane’s pullback speed would be adjusted such that the stones and the crane’s speed would be in an inversely proportional relation. Bigger the stone, slower the pullback speed and vice versa. Also proportions and the types of the stones would determine the points: bigger stone, greater points; gold stones provide points, regular stones don’t. These aspects I keep the same, however unlike the original game, in my version stones are not stationary but are moving from right of the screen to the left, because of reasons that I will elaborate on design methodology. The game ends when 15 of the stones past from right of the screen to the left without player hitting them.



Figure 3: Gold Miner Crane Different Angle

**Implementation Specification**

For this project I have used the following tools:

* Basys3
* One VGA Cable
* One VGA Compatible Monitor

While implementing the game I have decided to make crane and stones as square boxes, because it is easier and more consistent to write a collision detector using simple objects. For the movement of the crane, I assigned handpicked velocity values which gets updated at certain points. Since it would be hard and not so efficient to indicate where a square box would point while tilting the box around a vertical line, sense of trajectory is indicated with a dot which constantly oscillates horizontally between two points. For the input, only thing that the player should do is to press the middle button on the Basys3 board. When the button is pressed, indicator dot will stop oscillating and depending on where it stopped, new velocity values will be assigned to the crane. These values can be shown as:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1st Sec. | 2nd Sec. | 3rd Sec. | 4th Sec. | 5th Sec. | 6th Sec. | 7th Sec. |

Figure 4: Chart for Indicator Dot

If we imagine the below chart as number line, 4th Sec. would represent the 0 point. Hence 4th Sec. does a 0 degree angle with the vertical, 3rd Sec. does about 30 degrees, 2nd Sec. 45 degrees, 1st Sec. about 60 degrees and other sections mirror these values on different side of the vertical line. In each of these sections velocity values are handpicked and assigned. These velocity values determine in which angle will the crane travel and how fast it will reach its destination. If the crane collides with a stone, velocity values will be updated. After crane reaches its initial position, indicator dot will continue to oscillate where it left off. Game continues to function in this way until 15 stones pass, after that “GG” sign appears and the game ends.

**Project’s Design Methodology**

**Implementation of VGA Controller**

One obvious thing about designing a game is that it must be displayed. And in order for to display the game, first it is necessary to code a VGA controller. Because it is impossible to generate an image in the monitor at one given time; what a VGA controller does is that, much like the seven segment display, picks a pixel of the monitor at a time, refreshes it and moves onto the other ones. Because I have chosen 640x480 resolution, I have gathered back porch, front porch and sync pulse values according to my resolution and calibrated the values to the code I have found for VGA controller. VGA controller utilizes two counters to between pixels. First it goes through all the horizontal pixels, in my case 800, and assigns ‘0’ to bp, fp , sp intervals and then increments the vertical pixel counter by one and repeats the process until it reaches the final vertical pixel line, in my case 521, when it does it also resets back to the starting vertical line to repeat the process. Then I take these pixel coordinate values and subtract the bp, fp, sp values to get nice values from 0 to 480 and 0 to 640. And then these two variables that indicate on which pixel of the screen that the program is working on, are distributed to the modules. After the modules are done with the coordinates, I relate the coordinates with the corresponding colour and send them as RGB outputs thorough VGA cable to the monitor.

**Implementation of Clock Divider**

In order to generate an image, VGA controller needs a clock, and since almost every aspect of the game is built around the VGA controller, we need a clock. Basys3 already has a built-in clock but, built-in clock’s refresh rate is too fast for the 640x480 resolution that I have used, so I needed to slow it down. This module defines two variables named counter and temp and at every two clock cycles, at the rising edges of the clock, counter increments itself. After the counter reaches 2, temp equates itself to not temp and temp variable is assigned to the output as a clock. With this method clock changes after the counter reaches 2, instead of at every clock cycle. Every clock used in this project is this clock output, temp.

**Implementation of Button State Machine**

Whenan input is provided with the help of center button of Basys3, no input signal should be given to the other modules again until the crane hits some object and returns to its original position. For this reason we need a state machine, so that even if an input is provided from the button, crane will not stop its motion and start again from the start. To achieve this I have designed a state machine with two states. These are “first” and “second”. “First” state waits for the button to be pressed and after it is pressed it assigns the “btn” variable the value of ‘1’, while keeping the “btn” variable ‘1’ it goes to the “second” state. At the second state, state checks whether the crane has returned to its initial position and crane’s motion is upwards or not, if crane is at its initial position and crane is moving upwards then “btn” turns to ‘0’ and state goes back to first. This means that “btn” variable always stays as ‘1’ while the crane is moving and the initial variable is just a simple variable that turns to ‘1’ when the coordinates of the crane are the original ones that are predetermined. The reason why I check the crane’s motion as upwards or downwards is because, the initial variable is also equal to ‘1’ at the moment when the button is pressed. This means that as soon as the system goes to the second state it will go back to the first state without moving, which is an undesirable situation. So I also check if the crane is returning or just started to move, in order for it to work properly.

**Implementation of Draw Module**

Draw module allows me to assign the components on the screen to the RGB output with their colour attributed to them. I allocated each of the elements on the screen a draw attribute. Additionally this allows me to display the elements when I want and disable the elements when I don’t want. For example when the crane hits one of stones, stone should disappear until it is regenerated at the right edge of the screen, in the coming parts I will explain how I did this, and with draw attribute I simply disable until it is right time to display them again. Also this allows me to detect the collisions which I will explain next.

**Implementation of Collision Detectors**

There are multiple collision detectors because detection serves multiple roles in this design. First and simplest collision detector checks all the cases of collision. This collision type is used in the vertical motion tracker and the velocity updater module. Second type of collision is stone collision detector. This collision type is used for the score calculation and velocity updater module, as each stone has a different score value and as each stone has their own weight, their collision detection should be separate. These codes simply check if there is any pixel coordinate value which has both the crane\_draw and stone1\_draw(stone2\_draw, stone3\_draw…) attributes are set to ‘1’ and crane\_down is ‘1’. Crane must travel downwards because if this condition were to be removed, that would mean that when the crane bounced of one the edges and on its way hit some of the stones, collision would turn to ‘1’, which is an undesired situation because player didn’t directly aimed for the stone. If so, this means that there is a collision because two elements are on the same pixel and crane is traveling downwards, so collision is assigned the value of ‘1’.

**Implementation of Vertical Motion Tracker**

The crane’s vertical motion mentioned in previous sections is determined by a simple two state, state machine. For both of the states “btn” variable must be equal to ‘1’ and at the initial state, code checks whether there is a collision occurring or not and at the second state, it checks whether the crane is at the initial or not. So the way the state machine works is like following. Initial condition of “btn” being equal to ‘1’ means that the crane is in motion and when the crane is in motion because the initial state of the vertical motion is downwards, it will start to go down. After the collision occurs, it should start go upwards. If the collision is detected “crane\_down” variable is assigned the value of ‘0’ and state changes to second state. In this state when the crane is at the initial crane\_down is changed to ‘1’ and state is back to the first state. The variable crane\_down=’1’ basically means that the crane is going down. The crane\_down variable determines if the crane is going or returning and I will bring it up later in the velocity updater module section.

**Implementation of Indicator Dot**

One new variable that comes in to play that I haven’t mentioned before is new\_frame. The new\_frame variable is integer from 0 to 3, and is incremented by one each time the vertical line goes from 521 to 0. What this means is that each time the screen is fully refreshed the counter goes up, and when the counter reaches 3, that I will do the some of the processes below and this process. The reason being is that, even though I have slowed down the clock it is still too fast. If I try to display the moving of the crane or stones or the indicator dot, it is too fast for human eye to catch. So I added additional condition that only after every 3 refreshes that the coordinates are updated. Indicator dot also uses a three state, state machine process to determine the position of the dot. At the first state bar\_counter increments itself by one until it reaches 170 and the right\_motion variable is ‘1’. Then it goes to the second state where it subtracts itself by one until it reaches to 0 and the right\_motion variable is ‘0’. This cycle will continue if no input signal is given. State goes to the third state only if the “btn” goes to ‘1’. Then it will remain on the third state until “btn” turns to ‘0’, meaning that the motion is over. If the right\_motion variable is ‘1’, meaning that the indicator dot was going to the right before the state changed to third state, state will change to the first state; If the right\_motion variable is ‘0’, meaning that the indicator dot was going to the left before the state changed to third state, state will change to the second state. All these operations happen it the new\_frame is equal to 3, because indicator dot needs to be visible.

**Implementation of Velocity Updater**

Velocity updater is the longest code in my design. Essentially it is a one big state machine that controls all of the crane’s velocity values. State machine’s initial state works with the value of the indicator dot at the time of button press and according to that value assigns the corresponding velocity values. Indicator dot’s coordinates indicate where the crane should go for example, if we take the 0 coordinate as the most left coordinate of indicator dot; between 0 and 25 coordinates, crane’s horizontal velocity will be assigned -6 and vertical velocity will be assigned 4. Between 25 and 50, crane’s horizontal velocity will be assigned -6 and vertical velocity will be assigned 6 and between 50 and 85, crane’s horizontal velocity will be assigned -4 and vertical velocity will be assigned 6. What these values mean is that; ratio of horizontal to vertical value determines the angle of motion, while their Pythagoras summation gives the magnitude of velocity. I tried to choose these values such that it would emulate the sense of angle inclination with relation to indicator dot, while trying to keep their velocity equal. In addition to that when a collision with a stone occurs, other states come in play. Collision with the edges do not change the state because when a collision with an edge occurs speed values stay the same. I will mention why there is a minus sign in the horizontal velocities and how I add these velocity values in the next section. If the collision with a big stone occurs, velocity values are reduced by 66% and if the values were 6 and 4, they are set to 2 and 1. Collision with small stones however reduce the speed by 33% percent and these have their separate states. So there is a total of 15 state; one initial state for assigning velocity values based on the indicator dot’s position to give sense of angle, and as there are 7 intervals (3 for the left side, 3 for the right side and 1 for the middle) there are 7 states for big stone collision velocities and 7 states for small stone collision velocities. The values stay slowed down until the crane returns to its initial position. Since velocity updates should happen immediately after a collision occurs or the button is pressed, we do not need to do these processes using new\_frame variable.

**Implementation of Addition of Velocities to Coordinates**

Unlike the previous section, when the coordinates are updated we need to do this with the condition of new\_frame=3 as the velocity should be visible. The way the code creates the sense of velocity is that at every clock cycle where new\_frame=3 it adds the velocity values provided from velocity updater to the coordinates of the crane. So with this way it creates the illusion of a moving object. But it doesn’t always add up; for the case of crane going back after a collision, code should decrement from the pixel position of the crane. With this method incrementing the horizontal pixel values would mean to go to right and decrementing to left, increasing the vertical pixel values would mean to go down and decrementing to up. Hence the minus sign of horizontal velocity values when the indicator dot is between 0 and 85. Crane should go to the left at those values, so code adds coefficients with minus sign. The way that the code determines to whether increment or decrement is via the variable crane\_down, detecting if crane is going or returning.

**Implementation of Stone Generator**

Again with the help of new\_frame variable I start to decrement from one counter value. This counter value is determined by the horizontal pixel(640) and plus the width of the stone(40 if small, 65 if big) and the counter value is assigned to the horizontal pixel coordinate of the stones and sent to the draw module. Code starts with the initial stone, stone1, and at every new\_frame=3 code decrements the counter of stone1 and if the horizontal pixel value is equal to 450, sets the stone2 value to 640 plus the width of the stone2. All stone values get decremented all the time but the initial values of the stone counters are beyond the scope of the screen, so they do not get displayed, except stone1. Stone1 triggers the stone2 and after that stone2 sets the value of stone3 and the cycle continues. The integer 450 is chosen because this value allows me to display three stones in the screen at most, which I found most suitable for the challenge of the game. Also 640 plus the width of the stone is chosen because when the stone is set by the stone before it, it needs to flow continuously. So 640, width of the screen, and plus width of the stone means that it will flow without any delay.

**Implementation of Pseudo Random Generator**

I needed to generate random numbers because I wanted the order of the stones to be random. But since VHDL language is based on the working principles of electric components and there is hardly ever random number generating component, I struggled to find a method to do so. I decided to add to some unsigned number and take the bits of it. At every clock unsigned number gets incremented by one and at the moment of clock generation, third bit is assigned to determine the stone’s colour and second bit for the size. Because every moment of stone generation corresponds to the same clock time, pattern of the stones stay the same every time the game is restarted, hence the name pseudo, but pattern in itself is random.

**Implementation of Blasted Stones State Machine**

Last thing I coded on the project was another state machine. This state machine enables or disables the stones that are in the display. After the stones are hit by the crane, stones should disappear until they are regenerated at the rightmost of the screen. As I previously mentioned, I do this by controlling stone’s draw module. I added another “and” condition to the draw state of the stones: stone1\_coll. When the stone is hit, state changes from its initial state where all of the stone\_coll variables are equal to ‘1’, to the one state which the corresponding stonex\_coll is ‘0’. If the but there are additional states. If the player is good at the game, player can blast more than one of the stones before they regenerate. In fact all of the stones could be blasted before they regenerate, therefore there must be total of 8 states; one for initial, three for the cases of one stone getting blasted, three for two stones getting blasted, and one for all of the stones getting blasted. If the counter of the stones signals the state, state changes to one of the previous states in which whose stone counter is signalling the state and while doing that stonex\_coll also changes to ‘1’, enabling the draw module of the stone. Also at every state transition blasted\_stone variable gets incremented by one because state transitions mean that there is a collision with a stone and then this variable is subtracted from the number of stones that are generated to find out how many of the stones have passed without blasting. Upon which activates the game over condition.

**Implementation of Score Calculator and Seven Segment Display**

Even though I tried to carefully separate the processes that needed to be done at the rising edge of the clock with the ones that were needed to be done when new\_frame was equal to 3, I encountered a problem in this process. I was sure that I had coded the seven segment display and score calculations right but the display showed wrong results. Only problem that I could think of was that even though I tried to make collision calculations immediate and update the crane and stone positions at every 3 refresh cycles, that delay of 3 cycles was disrupting my collision calculation and adding more points than intended. So I coded another state machine which would change its state right after the collision occurred and wouldn’t update the value until the crane was at the initial position. This solved the issue and state machine basically does the same thing with my initial flawed code. It increments the counter by 1 if the stone type is small golden stone or by 2 if the stone type is big and golden and does nothing when stone type is small and blue. Then if the counter reaches 10, it increments the tens digit by one and resets the ones digit counter. Then these values are sent to the seven segment display module as a std\_logic\_vector format. I converted to this format because in order to not lose time coding seven segment display from scratch, I have modified the one that I have found.

**Results**

Following figures will show that the code is working as intended and without problems.

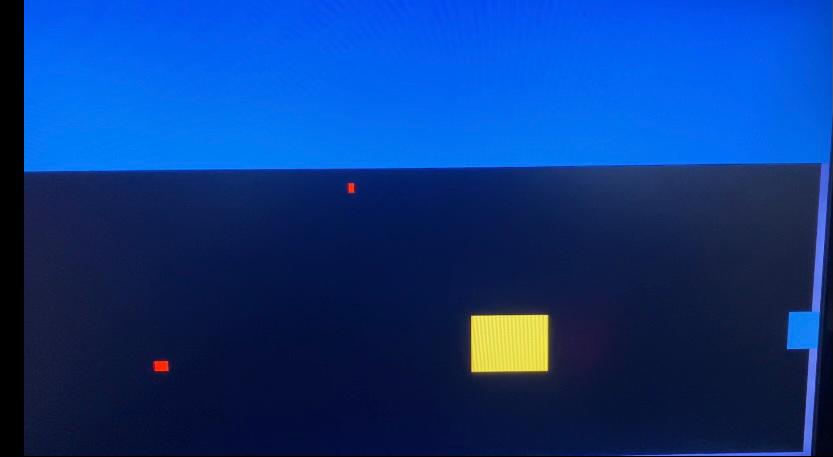
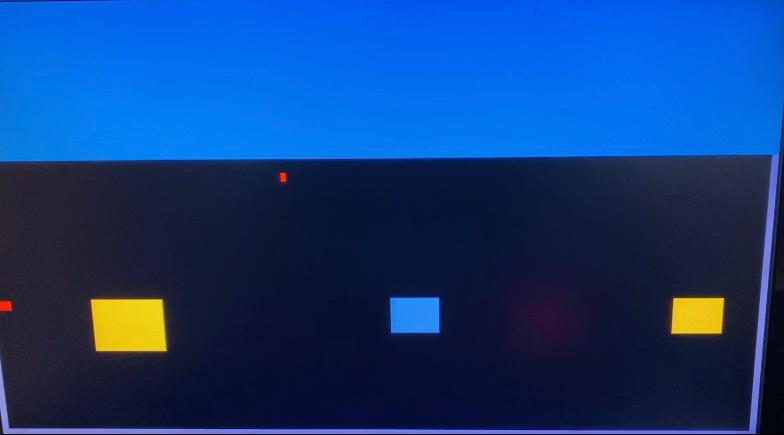


Figure 5: Indicator Dot Position 1 Figure 6: Indicator Dot Position 2

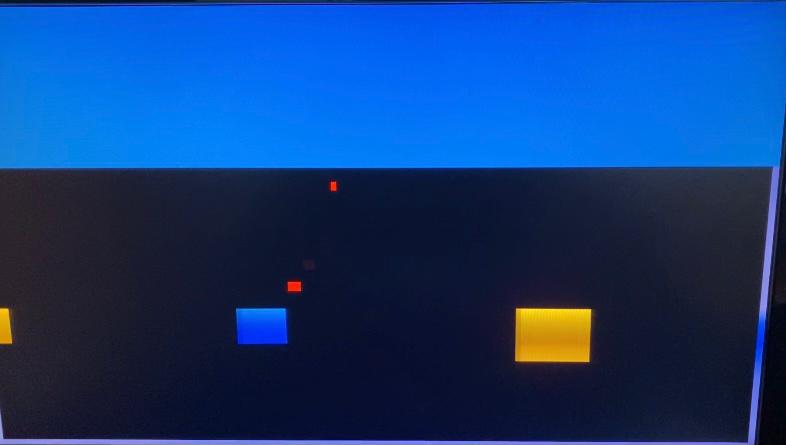


Figure 7: Indicator Dot Position 3 Figure 8: Indicator Dot Position 3(After Col.)



Figure 9: Indicator Dot Position 4 Figure 10: Indicator Dot Position 5

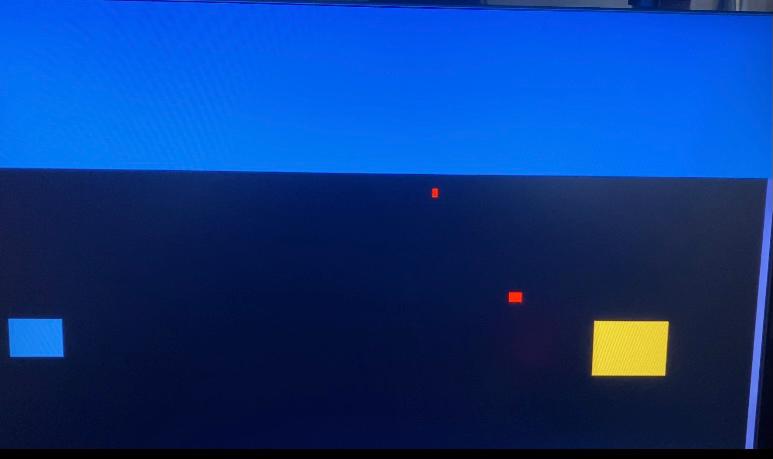


Figure 11: Indicator Dot Position 5(After Col.) Figure 12: Indicator Dot Position 6

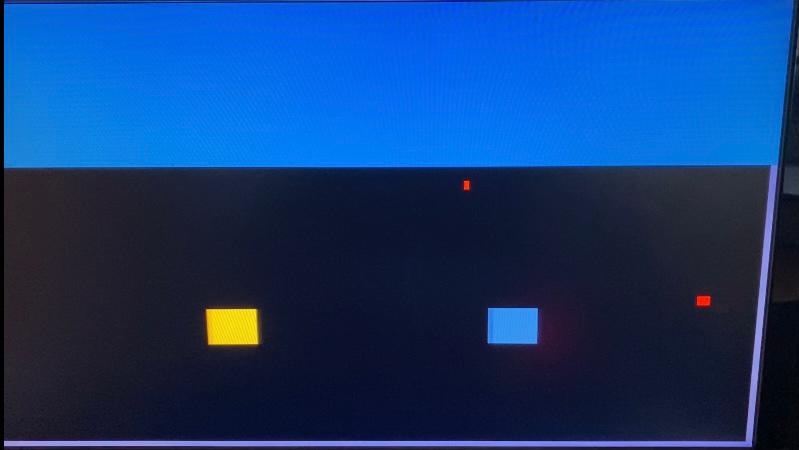


Figure 13: Indicator Dot Position 6(After Col.) Figure 14: Indicator Dot Position 7



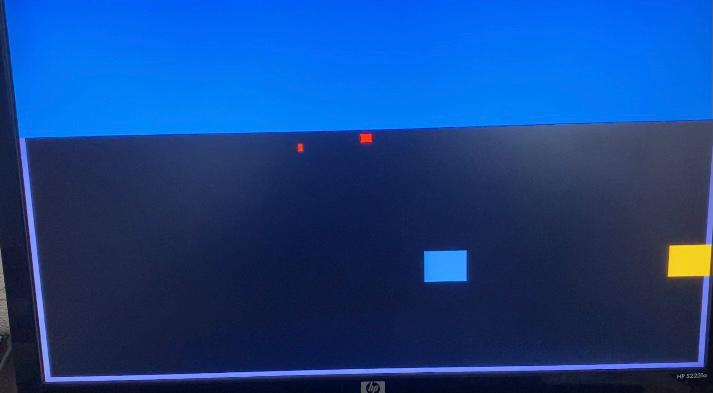


Figure 15: One Blasted Stone State Figure 16: Two Blasted Stones State



Figure 17: Three Blasted Stones State

In some of these figures it is hard to understand what is going on as it is a photograph of one specific moment of motion. In the video it is much clearer.

**Schematics**

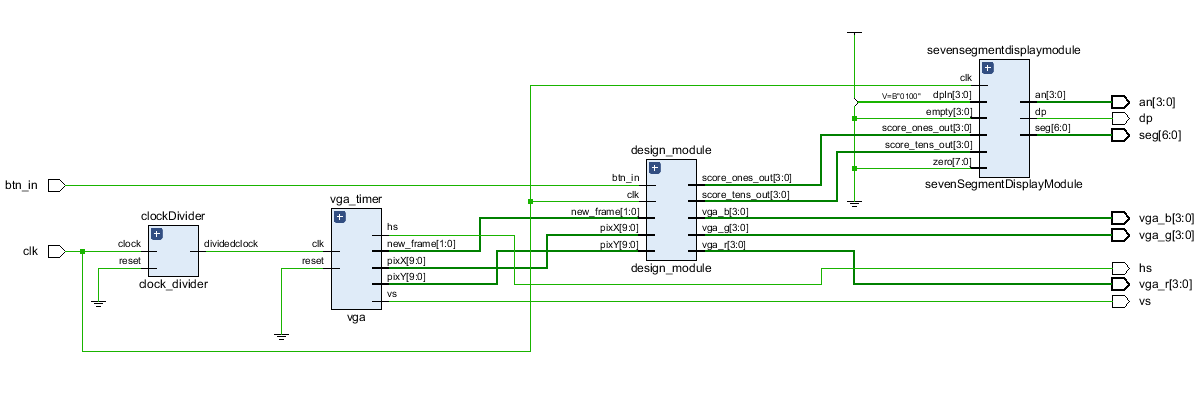


Figure 18: Top Module Schematic

This is the schematic of the top module. This shows how all the other components are connected. First the built-in clock of Basys3 gets modified in clock divider then connects to the other modules. Image display gets prepared in VGA controller then goes to the design module where every computation happens and all the RGB outputs are sent to the constraint. After that score calculated in design module goes to the seven segment display to get displayed.

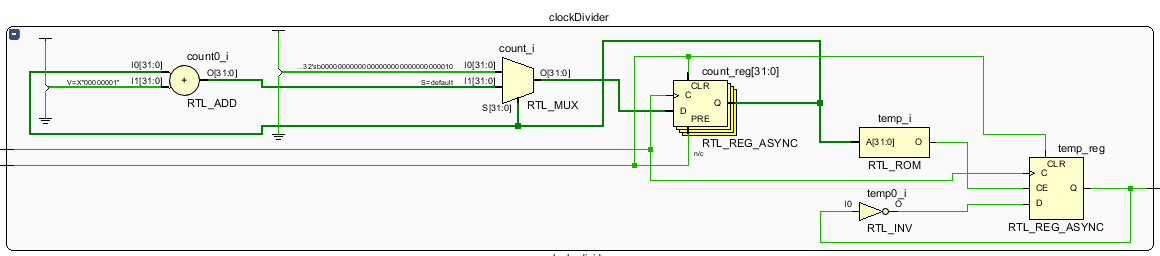


Figure 19: Clock Divider Schematic

Upper part of the circuit performs the necessary steps for restart signal but I don’t need the restart signal. I left it there when I took the code from outside sources since it doesn’t break the code. Other than that register takes the inverse of variable temp and takes it data to perform its task that I explained earlier.

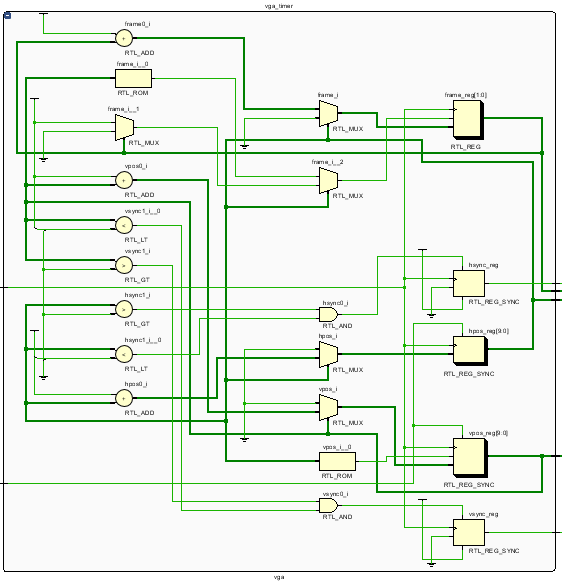


Figure 20: VGA Controller Schematic

First half of the circuit checks whether the predetermined conditions are met, if it reached 800 for horizontal and 521 for vertical, if not increments itself. Then the current position coordinate values are send to the design module, while other part of the circuit assigns ‘0’ to hp, fp, sp and ‘1’ to other pixels.

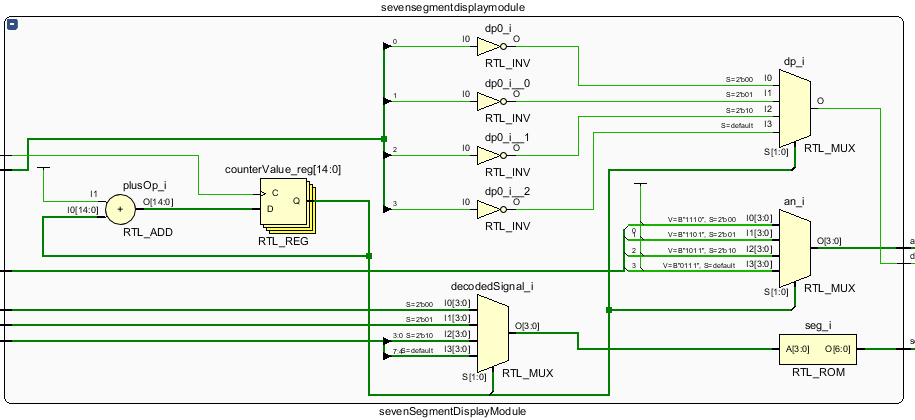


Figure 21: Seven Segment Display Schematic

The clock on the leftmost of the circuit determines which of the anodes will be refreshed and score provided from the design module is portioned to their respective LEDs.



Figure 22: Design Module Schematic

There are 6 state machines in this circuit and 3 other major processes that are done without state machines with their inputs and outputs interconnected to each other. Here I will attempt to explain them to the best extent that I can do. Firstly “random” variable will increment itself and at the moment of stone generation, bit values of it will be distributed to stone generation module with the help of multiplexers. Then the process of stone draw will commence and using greater than, less than, equal to components predetermined conditions will be checked and after that stone collision variable will be distributed with the help of multiplexers. Then the first type of collision detector will check the draw attribute of crane and the stones at the current pixel values using and gate to determine collision. Same is done for just stone draw and crane draw. Then the velocity updater commences, using comparators and “and” gates determines which paths will be activated and using multiplexers, assigns the determined values. The “and” gates check for the button input. All of these dribble down to one big multiplexer and then are sent to a register, this register gets activated at every 3 refresh cycles and sends these velocity values to the adders. These adders add the velocity values with the current pixel values and assigns them to the new pixel value. Finally all of the VGA values are gathered and are given to the RGB output.

**Problems Faced**

Like I have previously mentioned, one thing that bothered the functioning of the game for quite some time is the fact that I update the collisions and things that happen momentarily as such, at every clock cycle; whereas things concerning the velocities or things that should be visible to the human eye at every 3 refresh cycles. When I tried to change some of the velocity coefficient values this led to crane not stopping when it returned to its initial position and breaking the game. Because of that I had to limit the velocity values at most by 6 to 6. And countless of times I had to rewrite the modules where the new\_frame and clock operations collided.

One additional problem that I did not foresaw was one concerning the design of the game. Initially I planned to design the game somewhat like the original game, making the stone generate randomly on screen but stationary. But the original game had hand crafted levels and stones were placed intentionally, meaning my random generation of stones idea could ruin the gameplay. On top of that I haven’t fully figured out how I was going to implement the oscillation of the crane and thought that I could write an algorithm that determined the angle of throw as a function of time. This was impossible due to the fact that I have done all the horizontal and vertical pixel coordinate calculations using integers. There was no such algorithm to both equalize the velocity of the throw and adjust the angle of throw according to throw without using decimal numbers. So I had to change the velocities by hand and there were only a handful of values that I could use. Even the few values that I have used are problematic, velocity different between one throw of; 6 horizontal to 6 vertical, 6 horizontal to 4 vertical use visible. To preserve the angle I had to trade-off from the aspect of choosing their velocity values to be equal. So this approached limited me to pick just several intervals that the crane could travel, eliminating any sense of challenge and essentially fun in the game. So in order to increase the challenge I decided to make stones move.

**Conclusion**

When I was able to finish my project I was extremely satisfied by the fact that both I was able to complete such a grand scale of a project that I have not come close to attempting in the past and that I haven’t died. I was able to implement most of the things that I have learned in the previous lab works to my project and learn even more things and implement them. On top of that while using VHDL and analysing the elaborated designs, I was able to see how the codes that I have written corresponded to the real life electrical components. Additionally this project showed me that, how correctly labelling variables and adding comments on codes could potentially ease the workload on a project on such lengths.

**References**

GG Drawing, Seven Segment Display, Stone Generation Code and Clock Divider are taken from:

https://github.com/lukehsiao/FPGA\_Flappy\_Bird

VGA Controller is taken from:

https://github.com/ress/VHDL-Pong

**Appendix**

**Top Module:**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity top\_module is

Port (

clk, btn\_in: in std\_logic;

vga\_r, vga\_g, vga\_b: out std\_logic\_vector(3 downto 0);

an: out std\_logic\_vector(3 downto 0);

seg: out std\_logic\_vector(6 downto 0);

dp: out std\_logic;

hs, vs: out std\_logic

);

end top\_module;

architecture Behavioral of top\_module is

signal clk1: std\_logic;

signal rst: std\_logic:='0';

signal pixX: integer range 0 to 800;

signal pixY: integer range 0 to 521;

signal rgb: std\_logic\_vector(11 downto 0);

signal ground\_draw, sky\_draw: std\_logic;

signal new\_frame: integer range 0 to 2;

signal score\_tens\_out, score\_ones\_out: unsigned(3 downto 0);

signal dpIN: std\_logic\_vector(3 downto 0);

signal z: std\_logic\_vector(7 downto 0);

signal blankSegments: std\_logic\_vector(3 downto 0);

begin

vga\_timer: entity work.vga

port map(new\_frame=>new\_frame, clk=>clk1,reset=> rst, hs=>hs ,vs=>vs, pixX=>pixX, pixY=>pixY);

clockDivider: entity work.clock\_divider

port map(clock=>clk, dividedclock=>clk1, reset=>rst);

design\_module: entity work.design\_module

port map(new\_frame=>new\_frame, clk=>clk, btn\_in=>btn\_in, pixX=>pixX, pixY=>pixY,vga\_r=>vga\_r, vga\_g=>vga\_g, vga\_b=>vga\_b, score\_tens\_out=>score\_tens\_out, score\_ones\_out=>score\_ones\_out);

sevensegmentdisplaymodule: entity work.sevensegmentdisplaymodule

port map(clk=>clk,score\_tens\_out=>score\_tens\_out,score\_ones\_out=>score\_ones\_out, dpIN=>dpIN, empty=>blanksegments,seg=>seg, dp=>dp, an=>an, zero=>zero);

dpIN <= "0100";

zero <="00000000";

blankSegments <= "0000";

end Behavioral;

**Design Module:**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

use work.Constants.all;

entity design\_module is

Port (

clk, btn\_in: in std\_logic;

pixX: in integer range 0 to 800:=0;

pixY: in integer range 0 to 521:=0;

new\_frame: in integer range 0 to 3;

score\_tens\_out, score\_ones\_out: out unsigned(3 downto 0);

vga\_r, vga\_g, vga\_b: out std\_logic\_vector(3 downto 0)

);

end design\_module;

architecture Behavioral of design\_module is

constant default\_crane\_speed : integer :=1;

signal ground\_draw, sky\_draw, edges\_vertical\_draw, edges\_horizontal\_draw, crane\_draw, bar\_draw, game\_over\_draw: std\_logic;

signal ground, sky, edges, crane, bar, stone1, stone2,game\_over\_color: std\_logic\_vector(11 downto 0);

signal rgb: std\_logic\_vector(11 downto 0);

signal game\_over:std\_logic\_vector(11 downto 0);

signal pixelx: integer range -1 to 640:=0;

signal pixely: integer range -1 to 480:=0;

signal collision: std\_logic;

signal crane\_down: std\_logic:='1';

signal btn: std\_logic;

signal indicator: std\_logic:='0';

signal bar\_pos\_h: integer range -1 to 640:=235;

signal bar\_pos\_v: integer range -1 to 480:=180;

signal crane\_pos\_h: integer range -1 to 640:=315;

signal crane\_pos\_v: integer range -1 to 480:=170;

signal initial: std\_logic;

signal crane\_speed\_h: integer range -12 to 12;

signal crane\_speed\_v: integer range -12 to 12;

-- state machine signals for vertical motion designator

type state\_type is (a,b);

signal vertical\_state: state\_type;

-- state machine signals for button blocker

type stater is(first, second, third);

signal button\_blocker: stater;

-- state machine signals for bar counter

type b\_state is(alpha, beta, charlie);

signal bar\_state: b\_state;

signal bar\_counter: integer range 0 to 170 :=0;

signal right\_motion: std\_logic;

--state machine for velocity controller

type v\_state is(mono,a1,a2,b1,b2,c1,c2,d1,d2,e1,e2,f1,f2,g1,g2);

signal velocity\_control: v\_state;

-- state machine for collision holders

type stone\_collision is(stay,x,y,z,xy,xz,yz,xyz);

signal collide\_state: stone\_collision;

signal stone1\_coll,stone2\_coll,stone3\_coll: std\_logic:='0';

-- state machine signals for score counter

type state\_score is(detect, count);

signal counter\_state: state\_score;

-- signals for slowing down the velocity

signal counter\_bar: integer range 0 to 9:=0;

signal velocity\_factor: std\_logic;

-- signal for psuedo random numbers

signal random\_int: integer range 0 to 15:=0;

signal random: unsigned(3 downto 0):="0000";

-- signals for drawing the stones

signal stone1\_x: integer range 0 to 780:=640;

signal stone2\_x, stone3\_x: integer range 0 to 780:=780;

signal stone1\_y, stone2\_y, stone3\_y: integer range 0 to 480:=320;

signal stone1\_color, stone2\_color, stone3\_color: std\_logic;

signal stone1\_size, stone2\_size,stone3\_size: std\_logic;

signal stone1\_draw, stone2\_draw, stone3\_draw: std\_logic;

signal stonetype1\_draw, stonetype2\_draw, stonetype3\_draw, stonetype4\_draw: std\_logic;

signal stone1\_width, stone2\_width, stone3\_width: integer range -1 to 80:=0;

signal stone1\_collision, stone2\_collision, stone3\_collision, stone4\_collision: std\_logic;

signal stone1\_counter, stone2\_counter, stone3\_counter: integer range 0 to 1:=0;

signal stone1\_draw\_on, stone2\_draw\_on, stone3\_draw\_on: std\_logic;

-- counter signals

signal score\_tens,score\_ones: unsigned(3 downto 0):=(others=>'0');

-- gameover signals

type gameover\_s is (br,uh);

signal over\_button:gameover\_s;

signal generated\_stones: integer range 0 to 1000:=0;

signal blasted\_stones: integer range 0 to 1000:=0;

signal btn\_holder: std\_logic:='0';

begin

pixelx<=pixX-144;

pixely<=pixY-31;

ground <= "010000010000";

sky <= "001101111111";

edges<= "011101111111";

crane<= "111100100001";

bar<= "111100100001";

stone1 <="111111010001";

stone2 <="010001111111";

game\_over\_color<="111111111111";

bar\_draw <= '1' when (pixelx>= bar\_pos\_h and pixelx< bar\_pos\_h+4) and (pixely >= bar\_pos\_v and pixely< bar\_pos\_v+9) else '0';

crane\_draw <= '1' when (pixelx >= crane\_pos\_h and pixelx < crane\_pos\_h + 10) and (pixely >= crane\_pos\_v and pixely < crane\_pos\_v + 10) else '0';

edges\_horizontal\_draw<= '1' when ((pixelx>0 and pixelx<=640) and (pixely>474 and pixely<=480))else '0';

edges\_vertical\_draw <= '1' when (pixely>=165 and pixely<=474) and ((pixelx>0 and pixelx<6) or (pixelx>635 and pixelx<=640))else '0';

--ground\_draw <= '1' when (pixely>=165 and pixely<=474)and(pixelx>=6 and pixelx<=634) else '0';

sky\_draw <= '1' when (pixely<165 and(pixelx>0 and pixelx<=640)) else '0';

game\_over\_draw <= '1' when

((pixelx>=280 and pixelx<=310 and pixely>=215 and pixely<=220) or

(pixelx>=280 and pixelx<=285 and pixely>=215 and pixely<=265) or

(pixelx>=280 and pixelx<=310 and pixely>=260 and pixely<=265) or

(pixelx>=305 and pixelx<=310 and pixely>=235 and pixely<=265) or

(pixelx>=300 and pixelx<=310 and pixely>=235 and pixely<=245) or

(pixelx>=330 and pixelx<=360 and pixely>=215 and pixely<=220) or

(pixelx>=330 and pixelx<=335 and pixely>=215 and pixely<=265) or

(pixelx>=330 and pixelx<=360 and pixely>=260 and pixely<=265) or

(pixelx>=355 and pixelx<=360 and pixely>=235 and pixely<=265) or

(pixelx>=350 and pixelx<=360 and pixely>=235 and pixely<=245))

else '0';

process(clk)

begin

if rising\_edge(clk) then

if ((crane\_draw='1') and (edges\_horizontal\_draw='1' or edges\_vertical\_draw='1'or stonetype1\_draw='1' or stonetype2\_draw='1' or stonetype3\_draw='1' or stonetype4\_draw='1')) then

collision <='1';

else

collision <='0';

end if;

if crane\_draw='1' and stonetype1\_draw='1' then

stone1\_collision<='1';

else

stone1\_collision<='0';

end if;

if crane\_draw='1' and stonetype2\_draw='1' then

stone2\_collision<='1';

else

stone2\_collision<='0';

end if;

if crane\_draw='1' and stonetype3\_draw='1' then

stone3\_collision<='1';

else

stone3\_collision<='0';

end if;

if crane\_draw='1' and stonetype4\_draw='1' then

stone4\_collision<='1';

else

stone4\_collision<='0';

end if;

end if;

end process;

process(clk)

begin

if rising\_edge(clk) then

case collide\_state is

when stay =>

if stone1\_draw\_on='1' and crane\_draw='1' and crane\_down='1' then

if btn\_holder='1' then

blasted\_stones<= blasted\_stones+1;

end if;

stone1\_coll <='1';

collide\_state<=x;

elsif stone2\_draw\_on='1' and crane\_draw='1' and crane\_down='1' then

if btn\_holder='1' then

blasted\_stones<= blasted\_stones+1;

end if;

stone2\_coll <='1';

collide\_state<=y;

elsif stone3\_draw\_on='1' and crane\_draw='1' and crane\_down='1' then

if btn\_holder='1' then

blasted\_stones<= blasted\_stones+1;

end if;

stone3\_coll <='1';

collide\_state<=z;

else

stone1\_coll <='0';

stone2\_coll <='0';

stone3\_coll <='0';

collide\_state<=stay;

end if;

when x=>

if stone1\_x=1 then

collide\_state<=stay;

elsif stone2\_draw\_on='1' and crane\_draw='1' and crane\_down='1' then

if btn\_holder='1' then

blasted\_stones<= blasted\_stones+1;

end if;

stone2\_coll <='1';

collide\_state<=xy;

elsif stone3\_draw\_on='1' and crane\_draw='1' and crane\_down='1' then

if btn\_holder='1' then

blasted\_stones<= blasted\_stones+1;

end if;

stone3\_coll <='1';

collide\_state<=xz;

else

collide\_state<=x;

end if;

when y=>

if stone2\_x=1 then

collide\_state<=stay;

elsif stone1\_draw\_on='1' and crane\_draw='1' and crane\_down='1' then

if btn\_holder='1' then

blasted\_stones<= blasted\_stones+1;

end if;

stone1\_coll <='1';

collide\_state<=xy;

elsif stone3\_draw\_on='1' and crane\_draw='1' and crane\_down='1' then

if btn\_holder='1' then

blasted\_stones<= blasted\_stones+1;

end if;

stone3\_coll <='1';

collide\_state<=yz;

else

collide\_state<=y;

end if;

when z=>

if stone3\_x=1 then

collide\_state<=stay;

elsif stone1\_draw\_on='1' and crane\_draw='1' and crane\_down='1' then

if btn\_holder='1' then

blasted\_stones<= blasted\_stones+1;

end if;

stone1\_coll <='1';

collide\_state<=xz;

elsif stone2\_draw\_on='1' and crane\_draw='1' and crane\_down='1' then

if btn\_holder='1' then

blasted\_stones<= blasted\_stones+1;

end if;

stone2\_coll <='1';

collide\_state<=yz;

else

collide\_state<=z;

end if;

when xy=>

if stone1\_x=1 then

stone1\_coll <='0';

collide\_state<=y;

elsif stone2\_x=1 then

stone2\_coll <='0';

collide\_state<=x;

elsif stone3\_draw\_on='1' and crane\_draw='1' and crane\_down='1' then

if btn\_holder='1' then

blasted\_stones<= blasted\_stones+1;

end if;

stone3\_coll <='1';

collide\_state<=xyz;

else

collide\_state<=xy;

end if;

when yz=>

if stone2\_x=1 then

stone2\_coll <='0';

collide\_state<=z;

elsif stone3\_x=1 then

stone3\_coll <='0';

collide\_state<=y;

elsif stone2\_draw\_on='1' and crane\_draw='1' and crane\_down='1' then

if btn\_holder='1' then

blasted\_stones<= blasted\_stones+1;

end if;

stone2\_coll <='1';

collide\_state<=xyz;

else

collide\_state<=yz;

end if;

when xz=>

if stone3\_x=1 then

stone3\_coll <='0';

collide\_state<=x;

elsif stone1\_x=1 then

stone1\_coll <='0';

collide\_state<=z;

elsif stone2\_draw\_on='1' and crane\_draw='1' and crane\_down='1' then

blasted\_stones<= blasted\_stones+1;

stone2\_coll <='1';

collide\_state<=xyz;

else

collide\_state<=xz;

end if;

when xyz=>

if stone1\_x=1 then

stone1\_coll <='0';

collide\_state<=yz;

elsif stone2\_x=1 then

stone2\_coll <='0';

collide\_state<=xz;

elsif stone3\_x=1 then

stone3\_coll <='0';

collide\_state<=xy;

else

collide\_state<=xyz;

end if;

end case;

end if;

end process;

-- state machine for crane's vertical motion

process(clk, btn)

begin

if rising\_edge(clk) and btn='1' then

case vertical\_state is

when a =>

if collision='1' then

crane\_down<='0';

vertical\_state<=b;

else

vertical\_state<= a;

end if;

when b =>

if initial='1' then

crane\_down<='1';

vertical\_state<=a;

else

vertical\_state<=b;

end if;

end case;

end if;

end process;

-- state machine for the button

process(clk)

begin

if rising\_edge(clk) then

case button\_blocker is

when first =>

indicator<='0';

if btn\_in='1' then

btn<='1';

button\_blocker <= second;

else

button\_blocker<=first;

end if;

when second =>

if initial='1' and crane\_down='0' then

indicator<='1';

btn<='0';

button\_blocker <= third;

else

button\_blocker <= second;

end if;

when third =>

if btn\_in='0' then

indicator<='1';

button\_blocker <= first;

else

button\_blocker<= third;

end if;

end case;

end if;

end process;

-- state machine for the oscilation

process(new\_frame, btn, clk)

begin

if rising\_edge(clk) and new\_frame=3 then

case bar\_state is

when alpha =>

right\_motion <='1';

bar\_counter<= bar\_counter+1;

if bar\_counter= 170 then

bar\_state <= beta;

elsif btn='1' then

bar\_state <= charlie;

else

bar\_state <= alpha;

end if;

when beta =>

right\_motion <='0';

bar\_counter<= bar\_counter-1;

if bar\_counter=0 then

bar\_state <= alpha;

elsif btn='1' then

bar\_state <= charlie;

else

bar\_state <= beta;

end if;

when charlie =>

if btn='0' and right\_motion='1' then

bar\_state <= alpha;

elsif btn='0' and right\_motion='0' then

bar\_state <= beta;

else

bar\_state <= charlie;

end if;

end case;

end if;

end process;

bar\_pos\_h<= 235 + bar\_counter;

-- process for defining horizontal and vertical velocities

process(btn, clk,stone1\_collision,stone2\_collision,stone3\_collision,stone4\_collision)

begin

if rising\_edge(clk) and btn='1' then

case velocity\_control is

when mono =>

if (bar\_counter >= 0 and bar\_counter<22) then

if collision='0' then

crane\_speed\_h<= -4;

crane\_speed\_v<= 2;

velocity\_control<=mono;

elsif (stone1\_collision='1'or stone3\_collision='1') and crane\_down='1' then

velocity\_control<= a1;

elsif (stone2\_collision='1'or stone4\_collision='1') and crane\_down='1'then

velocity\_control<= a2;

end if;

elsif (bar\_counter >= 22 and bar\_counter<52) then

if collision='0' then

crane\_speed\_h<= -6;

crane\_speed\_v<= 6;

velocity\_control<=mono;

elsif (stone1\_collision='1'or stone3\_collision='1') and crane\_down='1'then

velocity\_control<= b1;

elsif (stone2\_collision='1'or stone4\_collision='1') and crane\_down='1' then

velocity\_control<= b2;

end if;

elsif (bar\_counter >= 52 and bar\_counter<72)then

if collision='0' then

crane\_speed\_h<= -3;

crane\_speed\_v<= 6;

velocity\_control<=mono;

elsif (stone1\_collision='1'or stone3\_collision='1')and crane\_down='1' then

velocity\_control<= c1;

elsif (stone2\_collision='1'or stone4\_collision='1')and crane\_down='1' then

velocity\_control<= c2;

end if;

elsif (bar\_counter >= 72 and bar\_counter<=98) then

if collision='0' then

crane\_speed\_h<= 0;

crane\_speed\_v<= 8;

velocity\_control<=mono;

elsif (stone1\_collision='1'or stone3\_collision='1')and crane\_down='1' then

velocity\_control<= d1;

elsif (stone2\_collision='1' or stone4\_collision='1')and crane\_down='1' then

velocity\_control<= d2;

end if;

elsif (bar\_counter > 98 and bar\_counter <=118) then

if collision='0' then

crane\_speed\_h<= 3;

crane\_speed\_v<= 6;

velocity\_control<=mono;

elsif (stone1\_collision='1'or stone3\_collision='1')and crane\_down='1' then

velocity\_control<= e1;

elsif (stone2\_collision='1'or stone4\_collision='1')and crane\_down='1' then

velocity\_control<= e2;

end if;

elsif (bar\_counter > 118 and bar\_counter <=148) then

if collision='0' then

crane\_speed\_h<= 6;

crane\_speed\_v<= 6;

velocity\_control<=mono;

elsif (stone1\_collision='1'or stone3\_collision='1')and crane\_down='1' then

velocity\_control<= f1;

elsif (stone2\_collision='1'or stone4\_collision='1')and crane\_down='1' then

velocity\_control<= f2;

end if;

elsif (bar\_counter > 148 and bar\_counter <=170) then

if collision='0' then

crane\_speed\_h<= 4;

crane\_speed\_v<= 2;

velocity\_control<=mono;

elsif (stone1\_collision='1' or stone3\_collision='1')and crane\_down='1' then

velocity\_control<= g1;

elsif (stone2\_collision='1'or stone4\_collision='1')and crane\_down='1' then

velocity\_control<= g2;

end if;

end if;

when a1 =>

crane\_speed\_h<=-4;

crane\_speed\_v<=2;

if initial='1' then

velocity\_control<=mono;

else

velocity\_control<=a1;

end if;

when a2 =>

crane\_speed\_h<=-2;

crane\_speed\_v<=1;

if initial='1' then

velocity\_control<=mono;

else

velocity\_control<=a2;

end if;

when b1 =>

crane\_speed\_h<=-4;

crane\_speed\_v<=4;

if initial='1' then

velocity\_control<=mono;

else

velocity\_control<=b1;

end if;

when b2 =>

crane\_speed\_h<=-2;

crane\_speed\_v<=2;

if initial='1' then

velocity\_control<=mono;

else

velocity\_control<=b2;

end if;

when c1 =>

crane\_speed\_h<=-2;

crane\_speed\_v<=4;

if initial='1' then

velocity\_control<=mono;

else

velocity\_control<=c1;

end if;

when c2 =>

crane\_speed\_h<=-1;

crane\_speed\_v<=2;

if initial='1' then

velocity\_control<=mono;

else

velocity\_control<=c2;

end if;

when d1 =>

crane\_speed\_h<=0;

crane\_speed\_v<=4;

if initial='1' then

velocity\_control<=mono;

else

velocity\_control<=d1;

end if;

when d2 =>

crane\_speed\_h<=0;

crane\_speed\_v<=2;

if initial='1' then

velocity\_control<=mono;

else

velocity\_control<=d2;

end if;

when e1 =>

crane\_speed\_h<=2;

crane\_speed\_v<=4;

if initial='1' then

velocity\_control<=mono;

else

velocity\_control<=e1;

end if;

when e2 =>

crane\_speed\_h<=1;

crane\_speed\_v<=2;

if initial='1' then

velocity\_control<=mono;

else

velocity\_control<=e2;

end if;

when f1 =>

crane\_speed\_h<=4;

crane\_speed\_v<=4;

if initial='1' then

velocity\_control<=mono;

else

velocity\_control<=f1;

end if;

when f2 =>

crane\_speed\_h<=2;

crane\_speed\_v<=2;

if initial='1' then

velocity\_control<=mono;

else

velocity\_control<=f2;

end if;

when g1 =>

crane\_speed\_h<=4;

crane\_speed\_v<=2;

if initial='1' then

velocity\_control<=mono;

else

velocity\_control<=g1;

end if;

when g2 =>

crane\_speed\_h<=2;

crane\_speed\_v<=1;

if initial='1' then

velocity\_control<=mono;

else

velocity\_control<=g2;

end if;

end case;

end if;

end process;

process(btn, clk, new\_frame)

begin

if rising\_edge(clk) then

if crane\_down='1' and new\_frame= 3 and btn='1' then

crane\_pos\_h <= crane\_pos\_h + crane\_speed\_h;

crane\_pos\_v <= crane\_pos\_v + crane\_speed\_v;

elsif indicator='1' then

crane\_pos\_h <=315;

crane\_pos\_v <=170;

elsif crane\_down='0' and new\_frame= 3 and btn='1' then

crane\_pos\_h <= crane\_pos\_h - crane\_speed\_h;

crane\_pos\_v <= crane\_pos\_v - crane\_speed\_v;

end if;

if crane\_pos\_h=315 and crane\_pos\_v=170 then

initial<='1';

else

initial<='0';

end if;

end if;

end process;

stone1\_draw\_on<= '1' when (stone1\_draw='1' and ((pixelx>= stone1\_x-stone1\_width and pixelx< stone1\_x) or (stone1\_x<=stone1\_width and pixelx< stone1\_x)) and (pixely>=stone1\_y and pixely < stone1\_y+stone1\_width)) else '0';

stone2\_draw\_on<= '1' when (stone2\_draw='1' and ((pixelx>= stone2\_x-stone2\_width and pixelx< stone2\_x) or (stone2\_x<=stone2\_width and pixelx< stone2\_x)) and (pixely>=stone2\_y and pixely < stone2\_y+stone2\_width)) else '0';

stone3\_draw\_on<= '1' when (stone3\_draw='1' and ((pixelx>= stone3\_x-stone3\_width and pixelx< stone3\_x) or (stone3\_x<=stone3\_width and pixelx< stone3\_x)) and (pixely>=stone3\_y and pixely < stone3\_y+stone3\_width)) else '0';

stone1\_draw<= '1' when (stone1\_x<=639 + stone1\_width and stone1\_x>0) else'0';

stone2\_draw<= '1' when (stone2\_x<=639 + stone2\_width and stone2\_x>0) else'0';

stone3\_draw<= '1' when (stone3\_x<=639 + stone3\_width and stone3\_x>0) else'0';

stonetype1\_draw <= '1' when (stone1\_draw='1' and stone1\_coll='0' and (stone1\_color='0' and stone1\_size='0') and ((pixelx>= stone1\_x-stone1\_width and pixelx< stone1\_x) or (stone1\_x<=stone1\_width and pixelx< stone1\_x)) and (pixely>=stone1\_y and pixely < stone1\_y+stone1\_width)) or

(stone2\_draw='1' and stone2\_coll='0' and (stone2\_color='0' and stone2\_size='0') and ((pixelx>= stone2\_x-stone2\_width and pixelx< stone2\_x) or (stone2\_x<=stone2\_width and pixelx< stone2\_x)) and (pixely>=stone2\_y and pixely < stone2\_y+stone2\_width)) or

(stone3\_draw='1' and stone3\_coll='0' and (stone3\_color='0' and stone3\_size='0') and ((pixelx>= stone3\_x-stone3\_width and pixelx< stone3\_x) or (stone3\_x<=stone3\_width and pixelx< stone3\_x)) and (pixely>=stone3\_y and pixely < stone3\_y+stone3\_width))

else '0';

stonetype2\_draw <= '1' when (stone1\_draw='1' and stone1\_coll='0' and (stone1\_color='0' and stone1\_size='1') and ((pixelx>= stone1\_x-stone1\_width and pixelx< stone1\_x) or (stone1\_x<=stone1\_width and pixelx< stone1\_x)) and (pixely>=stone1\_y and pixely < stone1\_y+stone1\_width)) or

(stone2\_draw='1' and stone2\_coll='0' and (stone2\_color='0' and stone2\_size='1') and ((pixelx>= stone2\_x-stone2\_width and pixelx< stone2\_x) or (stone2\_x<=stone2\_width and pixelx< stone2\_x)) and (pixely>=stone2\_y and pixely < stone2\_y+stone2\_width)) or

(stone3\_draw='1' and stone3\_coll='0' and (stone3\_color='0' and stone3\_size='1') and ((pixelx>= stone3\_x-stone3\_width and pixelx< stone3\_x) or (stone3\_x<=stone3\_width and pixelx< stone3\_x)) and (pixely>=stone3\_y and pixely < stone3\_y+stone3\_width))

else '0';

stonetype3\_draw <= '1' when (stone1\_draw='1' and stone1\_coll='0' and (stone1\_color='1' and stone1\_size='0') and ((pixelx>= stone1\_x-stone1\_width and pixelx< stone1\_x) or (stone1\_x<=stone1\_width and pixelx< stone1\_x)) and (pixely>=stone1\_y and pixely < stone1\_y+stone1\_width)) or

(stone2\_draw='1' and stone2\_coll='0' and (stone2\_color='1' and stone2\_size='0') and ((pixelx>= stone2\_x-stone2\_width and pixelx< stone2\_x) or (stone2\_x<=stone2\_width and pixelx< stone2\_x)) and (pixely>=stone2\_y and pixely < stone2\_y+stone2\_width)) or

(stone3\_draw='1' and stone3\_coll='0' and (stone3\_color='1' and stone3\_size='0') and ((pixelx>= stone3\_x-stone3\_width and pixelx< stone3\_x) or (stone3\_x<=stone3\_width and pixelx< stone3\_x)) and (pixely>=stone3\_y and pixely < stone3\_y+stone3\_width))

else '0';

stonetype4\_draw <= '1' when (stone1\_draw='1' and stone1\_coll='0' and (stone1\_color='1' and stone1\_size='1') and ((pixelx>= stone1\_x-stone1\_width and pixelx< stone1\_x) or (stone1\_x<=stone1\_width and pixelx< stone1\_x)) and (pixely>=stone1\_y and pixely < stone1\_y+stone1\_width)) or

(stone2\_draw='1' and stone2\_coll='0' and (stone2\_color='1' and stone2\_size='1') and ((pixelx>= stone2\_x-stone2\_width and pixelx< stone2\_x) or (stone2\_x<=stone2\_width and pixelx< stone2\_x)) and (pixely>=stone2\_y and pixely < stone2\_y+stone2\_width)) or

(stone3\_draw='1' and stone3\_coll='0' and (stone3\_color='1' and stone3\_size='1') and ((pixelx>= stone3\_x-stone3\_width and pixelx< stone3\_x) or (stone3\_x<=stone3\_width and pixelx< stone3\_x)) and (pixely>=stone3\_y and pixely < stone3\_y+stone3\_width))

else '0';

process(clk, random, new\_frame, stone1\_x, stone2\_x, stone3\_x)

begin

if rising\_edge(clk) then

if new\_frame=3 then

if stone1\_draw='1' then

stone1\_x<= stone1\_x-1;

end if;

if stone2\_draw='1' then

stone2\_x<= stone2\_x-1;

end if;

if stone3\_draw='1' then

stone3\_x<= stone3\_x-1;

end if;

end if;

if stone3\_x= 450 then

if btn\_holder='1' and new\_frame=3 then

generated\_stones<=generated\_stones+1;

end if;

stone1\_x <= 638 +stone1\_width;

if random(3 downto 2)="1" then

stone1\_color<='0';

elsif random(3 downto 2)="0" then

stone1\_color<='1';

elsif random(2 downto 1)="1" then

stone1\_size<='0';

elsif random(2 downto 1)="0" then

stone1\_size<='1';

end if;

end if;

if stone1\_x= 450 then

if btn\_holder='1' and new\_frame= 3 then

generated\_stones<=generated\_stones+1;

end if;

stone2\_x <= 638 +stone2\_width;

if random(3 downto 2)="1" then

stone2\_color<='0';

elsif random(3 downto 2)="0" then

stone2\_color<='1';

elsif random(2 downto 1)="1" then

stone2\_size<='0';

elsif random(2 downto 1)="0" then

stone2\_size<='1';

end if;

end if;

if stone2\_x= 450 then

if btn\_holder='1' and new\_frame=3 then

generated\_stones<=generated\_stones+1;

end if;

stone3\_x <= 638 +stone3\_width;

if random(2 downto 1)="1" then

stone3\_color<='0';

elsif random(2 downto 1)="0" then

stone3\_color<='1';

elsif random(1 downto 0)="1" then

stone3\_size<='0';

elsif random(1 downto 0)="0" then

stone3\_size<='1';

end if;

end if;

end if;

end process;

process(clk)

begin

if rising\_edge(clk) then

score\_ones\_out<=score\_ones;

score\_tens\_out<=score\_tens;

random\_int <= random\_int+1;

random <= to\_unsigned(random\_int,4);

if stone1\_size='1' then

stone1\_width<=60;

else

stone1\_width<=40;

end if;

if stone2\_size='1' then

stone2\_width<=60;

else

stone2\_width<=40;

end if;

if stone3\_size='1' then

stone3\_width<=60;

else

stone3\_width<=40;

end if;

end if;

end process;

-- state machine for calculating scores

process(clk)

begin

if rising\_edge(clk) then

case counter\_state is

when detect =>

if stone1\_collision='1' and crane\_down='1' then

score\_ones<=score\_ones + 1;

counter\_state<=count;

elsif stone2\_collision='1' and crane\_down='1' then

score\_ones<=score\_ones + 2;

counter\_state<=count;

elsif score\_ones=10 then

score\_ones<=(others => '0');

score\_tens<= score\_tens+1;

counter\_state<=count;

elsif score\_ones=11 then

score\_ones<=(others => '0');

score\_ones<=score\_ones+1;

score\_tens<= score\_tens+1;

else

counter\_state<=detect;

end if;

when count =>

if initial='1' then

counter\_state<=detect;

else

counter\_state<= count;

end if;

end case;

end if;

end process;

-- calculating scores

rgb<= bar when bar\_draw='1' else

stone1 when stonetype1\_draw='1' else

stone1 when stonetype2\_draw='1' else

stone2 when stonetype3\_draw='1' else

stone2 when stonetype4\_draw='1' else

crane when crane\_draw='1' else

edges when edges\_vertical\_draw ='1' else

edges when edges\_horizontal\_draw ='1' else

sky when sky\_draw='1' else

(others=>'0');

game\_over <= game\_over\_color when game\_over\_draw='1' else

(others=>'0');

-- vga\_r<= rgb(11 downto 8);

-- vga\_g<= rgb(7 downto 4);

-- vga\_b<= rgb(3 downto 0);

process(clk)

begin

if rising\_edge(clk) then

case over\_button is

when br =>

if btn\_in='1' and btn\_holder= '0' then

over\_button<=uh;

else

over\_button<=br;

end if;

when uh =>

btn\_holder<='1';

over\_button<=uh;

end case;

end if;

end process;

process(clk)

begin

if rising\_edge(clk) then

if btn\_holder='1' and(generated\_stones-blasted\_stones)>=15 then

vga\_r<= game\_over(11 downto 8);

vga\_g<= game\_over(7 downto 4);

vga\_b<= game\_over(3 downto 0);

else

vga\_r<= rgb(11 downto 8);

vga\_g<= rgb(7 downto 4);

vga\_b<= rgb(3 downto 0);

end if;

end if;

end process;

end Behavioral;

**VGA Controller:**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

use work.Constants.all;

use IEEE.STD\_LOGIC\_ARITH.ALL;

entity vga is

port(

clk, reset: in std\_logic;

hs, vs: out std\_logic;

new\_frame: out integer range 0 to 3 :=0;

pixX: out integer range 0 to 800 :=0;

pixY: out integer range 0 to 521 :=0

);

end vga;

architecture arch of vga is

signal hsync: std\_logic;

signal vsync: std\_logic;

signal frame: integer range 0 to 3:=0;

signal hpos: integer range 0 to 800 :=0;

signal vpos: integer range 0 to 521 :=0;

begin

process(clk, reset)

begin

if (clk'event and clk='1') then

if hpos=800 then

hpos<=0;

vpos<= vpos+1;

if vpos=521 then

vpos<= 0;

frame<=frame+1;

else

vpos<= vpos+1;

end if;

else

hpos<= hpos+1;

if frame=3 then

frame<=0;

end if;

end if;

if (reset='1') then

hpos <= 0;

vpos <= 0;

end if;

if hpos>0 and hpos<97 then

hsync<='0';

else

hsync<='1';

end if;

if vpos>0 and vpos<3 then

vsync<='0';

else

vsync<='1';

end if;

end if;

end process;

hs<=hsync;

vs<=vsync;

new\_frame<=frame;

pixX <= hpos;

pixY <= vpos;

end arch;

**Clock Divider:**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.numeric\_std.ALL;

entity clock\_divider is

port ( clock,reset: in std\_logic;

dividedclock: out std\_logic);

end clock\_divider;

architecture behavioral of clock\_divider is

signal count: integer:=1;

signal temp : std\_logic := '0';

begin

process(clock,reset)

begin

if(reset = '1') then

count <= 1;

temp <= '0';

elsif(clock'event and clock = '1') then

count <= count + 1;

if (count = 2) then

temp <= NOT temp;

count <= 1;

end if;

end if;

dividedclock <= temp;

end process;

end behavioral;

**Seven Segment Display:**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity sevenSegmentDisplayModule is

generic(DIVIDING\_BITS: natural := 15);

port(

clk: in std\_logic;

score\_tens\_out, score\_ones\_out: in unsigned(3 downto 0);

dpIn: in std\_logic\_vector(3 downto 0);

zero: in std\_logic\_vector(7 downto 0);

empty: in std\_logic\_vector(3 downto 0);

seg: out std\_logic\_vector(6 downto 0);

dp: out std\_logic;

an: out std\_logic\_vector(3 downto 0)

);

end sevenSegmentDisplayModule;

architecture Behavioral of sevenSegmentDisplayModule is

signal counterValue: std\_logic\_vector(DIVIDING\_BITS-1 downto 0) := (others=>'0');

signal anodeSelect: std\_logic\_vector(1 downto 0);

signal decodedSignal: std\_logic\_vector(3 downto 0);

signal dataIn: std\_logic\_vector(15 downto 0);

begin

dataIN(7 downto 0)<=std\_logic\_vector(score\_tens\_out & score\_ones\_out);

dataIN(15 downto 8)<=zero;

process(clk)

begin

if (clk'event and clk='1') then

counterValue <= std\_logic\_vector(unsigned(counterValue) + 1);

end if;

end process;

anodeSelect <= counterValue(DIVIDING\_BITS-1 downto DIVIDING\_BITS-2);

with anodeSelect select

an <=

"111" & empty(0) when "00",

"11" & empty(1) & '1' when "01",

'1' & empty(2) & "11" when "10",

empty(3) & "111" when others;

with anodeSelect select

decodedSignal <=

dataIn(3 downto 0) when "00",

dataIn(7 downto 4) when "01",

dataIn(11 downto 8) when "10",

dataIn(15 downto 12) when others;

with anodeSelect select

dp <=

not dpIn(0) when "00",

not dpIn(1) when "01",

not dpIn(2) when "10",

not dpIn(3) when others;

with decodedSignal select

seg <= "1000000" when "0000", -- 0

"1111001" when "0001", -- 1

"0100100" when "0010", -- 2

"0110000" when "0011", -- 3

"0011001" when "0100", -- 4

"0010010" when "0101", -- 5

"0000010" when "0110", -- 6

"1111000" when "0111", -- 7

"0000000" when "1000", -- 8

"0010000" when "1001", -- 9

"0001000" when "1010", -- A

"0000011" when "1011", -- B

"1000110" when "1100", -- C

"0100001" when "1101", -- D

"0000110" when "1110", -- E

"0001110" when others; -- F

end Behavioral;

**Constraints:**

##Buttons

set\_property PACKAGE\_PIN U18 [get\_ports btn\_in]

set\_property IOSTANDARD LVCMOS33 [get\_ports btn\_in]

#set\_property PACKAGE\_PIN T18 [get\_ports btnU]

#set\_property IOSTANDARD LVCMOS33 [get\_ports btnU]

#set\_property PACKAGE\_PIN W19 [get\_ports btnL]

#set\_property IOSTANDARD LVCMOS33 [get\_ports btnL]

#set\_property PACKAGE\_PIN T17 [get\_ports btnR]

#set\_property IOSTANDARD LVCMOS33 [get\_ports btnR]

#set\_property PACKAGE\_PIN U17 [get\_ports btnD]

#set\_property IOSTANDARD LVCMOS33 [get\_ports btnD]

## LEDs

#set\_property PACKAGE\_PIN U16 [get\_ports {out1}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {out1}]

set\_property PACKAGE\_PIN W5 [get\_ports clk]

set\_property IOSTANDARD LVCMOS33 [get\_ports clk]

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports clk]

##VGA Connector

set\_property PACKAGE\_PIN G19 [get\_ports {vga\_r[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {vga\_r[0]}]

set\_property PACKAGE\_PIN H19 [get\_ports {vga\_r[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {vga\_r[1]}]

set\_property PACKAGE\_PIN J19 [get\_ports {vga\_r[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {vga\_r[2]}]

set\_property PACKAGE\_PIN N19 [get\_ports {vga\_r[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {vga\_r[3]}]

set\_property PACKAGE\_PIN N18 [get\_ports {vga\_b[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {vga\_b[0]}]

set\_property PACKAGE\_PIN L18 [get\_ports {vga\_b[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {vga\_b[1]}]

set\_property PACKAGE\_PIN K18 [get\_ports {vga\_b[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {vga\_b[2]}]

set\_property PACKAGE\_PIN J18 [get\_ports {vga\_b[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {vga\_b[3]}]

set\_property PACKAGE\_PIN J17 [get\_ports {vga\_g[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {vga\_g[0]}]

set\_property PACKAGE\_PIN H17 [get\_ports {vga\_g[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {vga\_g[1]}]

set\_property PACKAGE\_PIN G17 [get\_ports {vga\_g[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {vga\_g[2]}]

set\_property PACKAGE\_PIN D17 [get\_ports {vga\_g[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {vga\_g[3]}]

set\_property PACKAGE\_PIN P19 [get\_ports hs]

set\_property IOSTANDARD LVCMOS33 [get\_ports hs]

set\_property PACKAGE\_PIN R19 [get\_ports vs]

set\_property IOSTANDARD LVCMOS33 [get\_ports vs]

#7 segment display

set\_property PACKAGE\_PIN W7 [get\_ports {seg[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[0]}]

set\_property PACKAGE\_PIN W6 [get\_ports {seg[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[1]}]

set\_property PACKAGE\_PIN U8 [get\_ports {seg[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[2]}]

set\_property PACKAGE\_PIN V8 [get\_ports {seg[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[3]}]

set\_property PACKAGE\_PIN U5 [get\_ports {seg[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[4]}]

set\_property PACKAGE\_PIN V5 [get\_ports {seg[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[5]}]

set\_property PACKAGE\_PIN U7 [get\_ports {seg[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[6]}]

set\_property PACKAGE\_PIN V7 [get\_ports dp]

set\_property IOSTANDARD LVCMOS33 [get\_ports dp]

set\_property PACKAGE\_PIN U2 [get\_ports {an[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {an[0]}]

set\_property PACKAGE\_PIN U4 [get\_ports {an[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {an[1]}]

set\_property PACKAGE\_PIN V4 [get\_ports {an[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {an[2]}]

set\_property PACKAGE\_PIN W4 [get\_ports {an[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {an[3]}]